An Improved Autoassociative Network for Controlling Autonomous Robots

Charles Hand
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California, USA
chuck@brain.jpl.nasa.gov

Abstract: One promising approach to neural network controlled robotics is the use of autoassociative networks. These networks learn to move a "sensor and effector" vector through a plausible state-space. This approach is, however, hindered by the intrinsically inefficient nature of autoassociative networks. This paper outlines a novel approach that greatly increases the efficiency and resolution of associative networks, and has other implementation benefits as well.
Keywords: autonomous robots, walking robots, synaptic networks

Research Objective

Recently, there has been enormous interest in small insect-like robots. The Department of Defense would like to send dozens of "bugbots" to see what is over the next hill. NASA would like to drop a fleet of bugbots into the atmosphere of Mars. *Other* agencies would like to have bugbots inspect pipes (from the inside), or fly into buildings and then crawl from room to room looking for inhabitants.

A very natural and effective approach to neural network control of these small robots is the use of autoassociative networks to process vectors composed of current sensor and effector information. This approach has created some remarkably intelligent small robots like those developed by Erwin Baumann and David Williams of McDonell Douglas Aerospace [1] and Dario Flareano and Jseba Urzelai at the Swiss Federal Institute of Technology in Lausanne [2,3].

The use of autoassociative networks in robotics is hindered by shortcomings that are intrinsic to autoassociative networks themselves. For example, one measure of a network's ability to construct meaningful search space landscapes is the number of attractors that *can* form clearly delineated basins. The typical autoassociative net with N neurons *can* be expected to form only 0.15N clearly delineated basins even though there are 2^N patterns in the search space.

Other attributes of autoassociative nets that impede their use in robotics are caused by the difficulties inherent in training autoassociative networks, not the least of which is the need for a CPU and training programs. Typical training regimes tend to focus *on* learning only from good examples or only from bad examples when robots (like animals) need to learn from both. Furthermore, the stored program paradigm itself is brittle, intrinsically serial, prone to deadlocks, and degrades unpredictably [4]. The research described in this paper is aimed at freeing autoassociative networks from these problems.

Autoassociative Networks

Generally speaking, autoassociative networks are a set of neurons that are completely connected. Each neuron has input from all other neurons, and the output of each neuron goes to every other neuron. In some instantiations, neurons also output to themselves. The state of a neuron is Completely determined by the dot product of its inputs and its weights. Setting the weights sets the behavior of the network.

The neurons of an autoassociative network are usually thought of as comprising a row or vector. Time is a quantum phenomenon for (most) autoassociative networks in the sense that time proceeds in discrete steps or moments. At each moment of time, the row of neurons forms a pattern: Some neurons are firing, some are not firing. Hence the current state of an autoassociative network *can* be described with a single *binary* vector. As times goes by, the network changes this vector. Autoassociative networks move vectors over landscapes of possibilities.

If we look at the network from the point of view of a single weight we start to see some of the reasons that autoassociative networks are so inefficient. Consider a network with N neurons, and look at neuron M -- or more precisely look at weight W of neuron M. Learning consists of moving W to a number that is best for most patterns. At the end of training, all weights are fixed to reflect the tyranny of the majority of patterns. All the patterns that represent minorities (from a single weight's point of view) are ignored. The performance of the network would be improved if the fixed weights were replaced with something more dynamic.

Nexus Controlled Robots

A *nexus* is a "deeper" *[5]* autoassociative network. Where you would expect to find a weight in an autoassociative network, you find the output of a network in a nexus. In an autoassociative network, each neuron is connected to all (or most) of the other neurons in the net. If there are N neurons in the net, there will be (on the order of) N^2 connections between neurons. In a nexus, the number of weights will be N^j where $j>2$.

At each timestep a neuron is either firing (1) or not firing (0) depending on the current input from all the other neurons. Hence the matrix of weights connecting the neurons controls the movement of a vector (neuronal firing pattern) through N-space. The topography of trajectories through N-space is completely determined by the weight matrix. Learning consists of modifying the values of these weights.

Here are some of the ways in which a Nexus differs from an Autoassociative net:

1. A nexus uses synaptic networks [5,7] throughout. Autoassociative nets do not.
2. A nexus changes only the effector part of a vector. Autoassociative nets change all parts of the vector.
3. Top level Autoassociative weights are replaced by nets in nexi.
4. Nexus weights are binary and can be stored as memory bits. Autoassociative weights are numbers (stored in registers).
5. The only arithmetic operation in a nexus is 'majority).
6. A nexus has a simple learning algorithm that uses both 'good' and 'bad' examples. Autoassociative nets are (usually) trained using only negative examples.

A Bottom-Up Description of a Nexus

To see how a nexus works at the lowest level, consider a very small nexus that processes vectors of eleven elements. At the lowest level, there will be two rows of memory locations; each row can hold eleven bits. For ease of reference, we will represent the memory locations with letters of the alphabet:

```
a b c d e f g h i j k     (datazerosrow)
1 m n o p q r s t u v     (dataonesrow)
```

The vector 00000000000 consults memary locations
a, b, c, d, e, f, g, h, i, j, and k.
The vector 00001111100 consults memory locations
a, b, c, d, p, q, r, s, t, j, and k.

Consultation consists of looking at the specific locations and ascertaining if the majority of the locations contain the one bit (verses the zero bit). If the majority of the bits are ones, the value of the consultation is one, else the value of the consultation is zero.

Learning (from negative examples) consists of changing bits when a mistake is made. If it is known that the consultation should have given a one and instead gave a zero, then one or mote of the constituent bits is changed from zero to one. The treatment of false majorities is treated in a complementary manner. When the only high level feedback is that the judgment was in error, a random set of bits is changed. For additional information about the fine points of implementing the learning algorithm, see the references on synaptic networks [5,7].

Learning (from positive examples) is accomplished with the addition of two rows of 'creb' bits (the name comes from biology). One of these rows is associated with the zero data bits and the other is associated with the one data bits:

```
Z Z Z Z Z Z Z Z Z Z Z     (crebzerorow)
a b c d e f g h i j k     (data zero row)
l m n o p q r s t u v     (data one row)
N N N N N N N N N N N     (creb one row)
```

When a successful pattern evaluation is encountered, the creb bits that correspond to the pattern are set to one. Negative results may change the data rows, but positive results change only the creb rows. When a negative pattern indicates a change in a data bit, the corresponding creb bit is checked. If the creb bit is zero, the data bit is changed. If the creb bit is me, the creb bit is set to zero and the data is left unchanged. Hence, good examples tend to make bits 'sticky) in the sense that they resist future changes from bad examples. The preceding four rows of memory (two rows of data, two rows of creb) form the lowest level or terminal node of the nexus.

The next higher level up the nexus can be represented as the following three rows:

```
A B C D E F G H I J K     (zeros terminal nodes)
X X X X X X X X X X X     (targetvector)
L M N O P Q R S T U V     (onestenninalnodes)
```

The row of Xs holds the current vector. The upper row is a collection of terminal nodes for consultation by zero bits in the current vector; the lower row is a collection of terminal nodes for consultation by one bits in the current vector.

If the current vector is 00001111100, we have:

```
A B C D E F G H I J K     (zerosterminalnodes)
0 0 0 0 1 1 1 1 1 0 0     (target vector)
L M N O P Q R S T U V     (ones terminal nodes)
```

So to generate the target vector 00001111100, we consult the terminal node A for the leftmost node, the terminal node B for the next bit, and continue through C, D, P, Q, R, S, T, J, and K. Each terminal node is consulted using 00001111100 and each terminal node produces one bit of the new vector. This deepening process can be extended to any number of levels but there will be an exponential growth in the number of total bits.

A Top-Down Description of a Nexus

Although we refer to the set of weights that controls vector updating in the nexus as a matrix, it is always more complex than a single matrix. In the simplest possible nexus, vector updating is controlled by two matrices: a zero matrix and a ones matrix. In actual practice, each element of the zero matrix and each element of the ones matrix are (often) replaced by another level of synaptic net [5]. Therefore the elements of the top-level matrices are **not** fixed, but computed. This process of replacing a weight with a net is how simple synaptic networks 'snap together' to farm more complex networks. From a traditional artificial neural network perspective, the weights of an autoassociative net are being replaced with perceptrons.

When we descend to the lowest level and reach the terminal node we find bits in memory locations instead of numbers in registers as we would expect to find at the lowest level of autoassociative nets. The only form of arithmetic performed on a set of bits is the 'majority' function. The majority function returns one if the majority of bits in the input set are ones, and the majority function returns zero otherwise. These ones and zeros may be passed up to higher-level majority functions. Computing evaluations as the majority of bits has two practical high level ramifications: minimizing memory requirements and eliminating the need for a CPU [6].

An Example of Nexus Controlled Behavior

Gait control in hexapod walking robots is an excellent application of nexus control. There are three basic hexapod gaits (see Figure 1): bi-tripodal (pretty stable, pretty fast), side-to-side (unstable, very fast), and caterpillar (very stable, quite slow).

Figure 1 shows the three basic modes of locomotion.

The open circles represent raised feet and the filled circles represent feet that are in contact with the surface over which the hexapod is moving. These three gaits are the results of three different schemas [8]. We trained both nexi and ordinary autoassociative networks on all three gaits. The standard autoassociative networks could be preset to any (but not all) of the gaits. However, training the gaits with autoassociative networks proved to be very difficult without the use of auxiliary techniques such as simulated annealing and reinforcement learning -- techniques which introduced unnecessary complexity and superfluous hardware.

Figure 2 shows the errors during learning of all three gaits using a different single-layer nexus for each gait. Training sessions included (but were not limited to) positive feedback for forward progress and negative feedback for falling down.

The training of the three individual one-layer nets (Figure 2) *started* with random leg positions. The nexus was trained from whatever position the last move left the hexapod in. The three curves shown in Figure 2 are typical. The training was always successful no matter what the starting position.

Figure 3 shows the results for one deep nexus learning all three gaits, with the three gaits being reinstated every 10 steps. The learning time required by the deep nexus (Figure 3) was less than the total time required by the three single-layer nexi combined, but greater than the time required by the worst of the three single-layer nexi (Figure 2). This seems reasonable since the single-layer nexi were trained in parallel, and the deeper nexus required approximately 12 times as many weights as the smaller nexi. This finding is consistent with the findings in many areas of neural network research. Many small nets outperform one large net. [4,6]

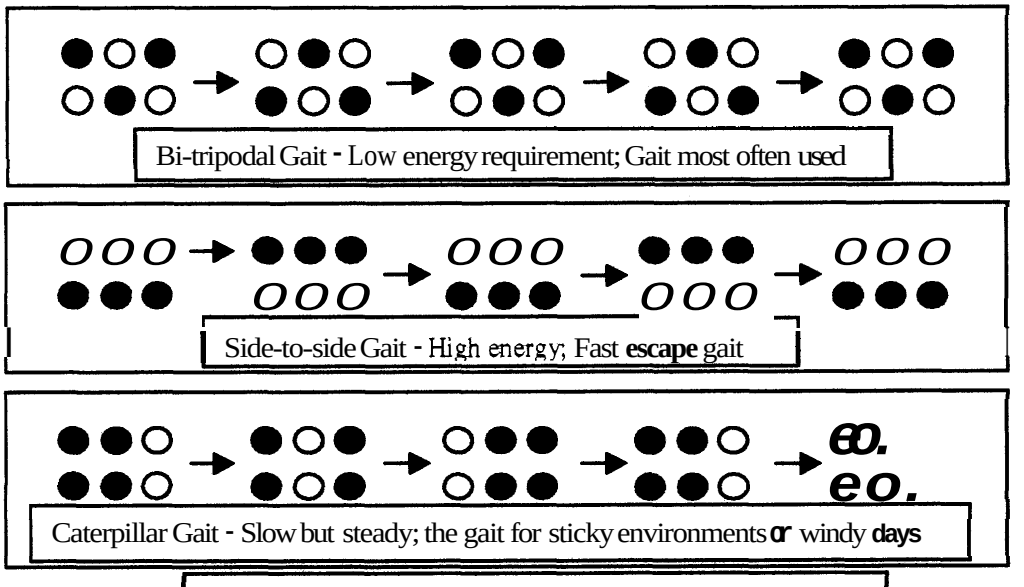

Figure 1. The three basic hexapod gaits

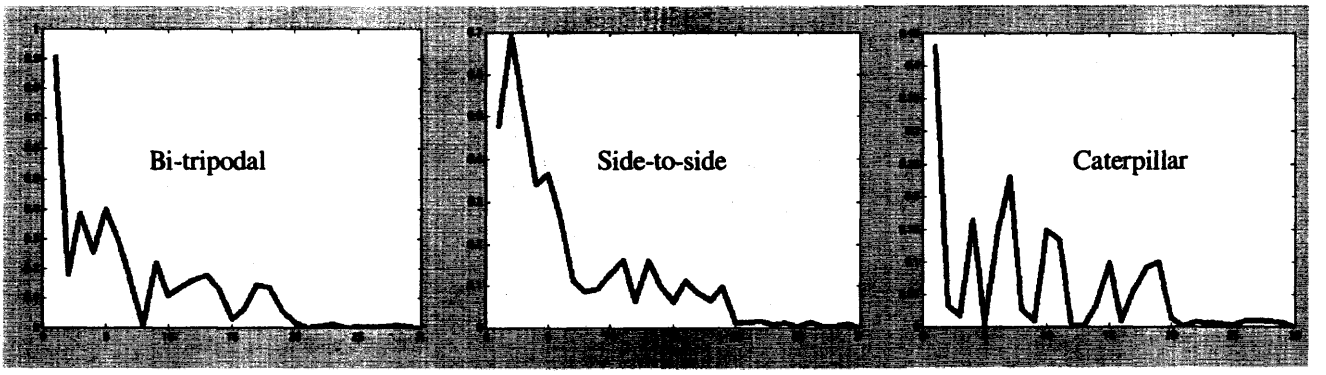

Figure 2. Each small nexus learning one gait

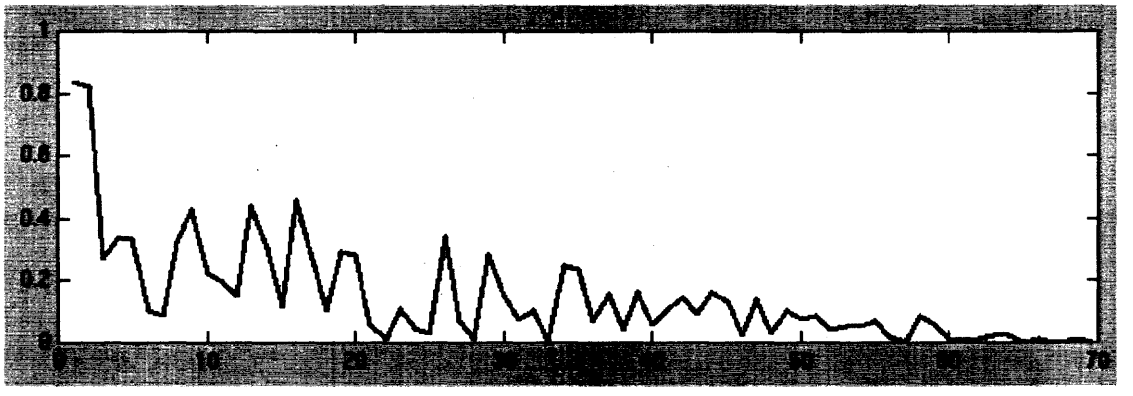

Figure 3. A deeper nexus learning all three gaits

Acknowledgement

The research described in this paper was performed at the Center for Integrated Space Microsystems, Jet Propulsion Laboratory, California Institute of Technology and was sponsored by the National Aeronautics and Space Administration.

References

[1] E. W. Baumann, and D. L. Williams, "Stochastic Associative Memory" in "The Science of Artificial Neural Networks 11", SPIE Proceedings vol. 1966, pp. 132-139,1966

[2] D. Floreano, and F. Mondada, "Evolutionary Neurocontrollers for Autonomous Mobile Robots", Neural Networks, vol. 11, pp. 1461-1478, 1998

[3] J. Urzelai, J. Floreano, M. Dorigo, and M. Colombetti, "Incremental Robot Shaping", Connection Science, vol. 10, pp. 341-360, 1998

r41 R. M. Golden, "Mathematical Methods for Neural Network Analysis and Design," MIT Press, 1994

[5] C. Hand, "A Pliant Synaptic Network for Signal Analysis", The International Conference on Mathematical and Engineering Techniques in Medicine and Biological Sciences, vol. 1, pp. 275-281, METMBS Press, 2000

[6] M. Spitzer, "The Mind Within the Net", MIT Press, 1999

[7] C. Hand, "Genetic Nets," Proceedings 1997 IEEE Conference on Genetic Programming, Stanford University, vol. 2, pp. 3541, 1997

[8] M. A. Arbib, "Handbook of Brain Theory and Neural Networks", MIT Press, 1995